

CSSE 132 – Introduction to Computer Systems
 Rose-Hulman Institute of Technology
 Computer Science and Software Engineering Department

ARM vs. C — Some examples

1 Simplified Examples

These examples assume all the variable data are stored in registers. This is not usually the case, but this helps illustrate basic C structures.

1.1 main function

```
1 int main() {
2     int x = 13;
3     int y = 14;
4     return x + y;
5 }
```

```
1 main:
2     mov r2, #13
3     mov r3, #14
4     add r0, r2, r3
5     bx lr
```

1.2 if statement

```
1 x = 10;
2
3 if ( x > 0 ) {
4
5     x = x - 1;
6 }
```

```
1     mov r3, #10    @ r3 holds 'x'
2 .BEGINIF:
3     cmp r3, #0
4     ble .ENDIF    @ x>0 = !(x<=0)
5     sub r3, r3, #1
6 .ENDIF:
```

1.3 if/else statement

```
1 x = 10;
2
3 if ( x > 0 ) {
4
5     x = x - 1;
6 } else {
7     x = x + 1;
8 }
9
```

```
1     mov r3, #10    @ r3 holds 'x'
2 .BEGINIF:
3     cmp r3, #0
4     ble .ELSE    @ x>0 = !(x<=0)
5     sub r3, r3, #1
6     b .ENDIF
7 .ELSE:
8     add r3, r3, #1
9 .ENDIF:
```

1.4 do/while loop

```

1 x = 10;
2 do {
3     x = x - 1;
4 while ( x > 0 );

```

```

1     mov r3, #10      @ r3 holds 'x'
2 .LOOP:
3     sub r3, r3, #1
4     cmp r3, #0
5     bgt .LOOP
6 .ENDLOOP:

```

1.5 while loop

```

1 x = 10;
2
3 while ( x > 0 ) {
4
5     x = x - 1;
6 }

```

```

1     mov r3, #10      @ r3 holds 'x'
2 .LOOP:
3     cmp r3, #0
4     ble .ENDLOOP
5     sub r3, r3, #1
6     b .LOOP
7 .ENDLOOP:

```

There's another way to compile the same while loop. This way looks more like the do-while translation to assembly:

```

1     mov r3, #10      @ r3 holds 'x'
2     b .TEST
3 .LOOP:
4     sub r3, r3, #1
5 .TEST:
6     cmp r3, #0
7     bgt .LOOP
8 .ENDLOOP:

```

1.6 for loop

There's lots of extra space in this for loop. Usually you'd write the for statement like this:

for (x = 10; x > 0; x--) { ... } but to better see how the bits of C map to ARM, spaces are added:

```

1 y = 0;
2 for ( x = 10;
3
4     x > 0 ;
5
6     x--      ) {
7     y = y + x;
8 }

```

```

1     mov r2, #0        @ r2 holds 'y'
2     mov r3, #10       @ r3 holds 'x'
3 .LOOP:
4     cmp r3, #0
5     ble .ENDLOOP      @ stop if !(x>0)
6     sub r3, r3, #1     @ x--
7     add r2, r2, r3     @ y=y+x
8     b .LOOP
9 .ENDLOOP:

```

2 Load/Store Examples

These examples now assume variables are assigned a memory location. This means they are loaded and stored as necessary.

2.1 set elements in array

Assume `x` is an array and its address is stored in `r4`:

```

1 x[0] = 40;
2
3
4 x[1] = 30;
5
6
7 x[2] = x[0] + x[1];

```

```

1  mov r0, #40
2  str r0, [r4] @ r4 is addr of x
3
4  mov r0, #30
5  str r0, [r4, #4]
6
7  ldr r0, [r4]      @ x[0]
8  ldr r1, [r4, #4] @ x[1]
9  add r0, r0, r1    @ x[0]+x[1]
10 str r0, [r4, #8]

```

2.2 Local variables stored on the stack

When you declare variables in a C function, it makes space on the stack by moving `sp`, then assigns the variables locations there (much like entries in an array). Notice that `x = y`; requires both a load from memory and a store into memory since both variables are stored in memory!

In this example `x`, `y` and `tmp` are local variables stored on the stack:

```

1 void swap()
2 {
3
4     int x = 1;
5
6
7     int y = 16;
8
9
10    int tmp = x;
11
12
13    x = y;
14
15
16    y = tmp;
17
18
19 }

```

```

1 swap:
2     sub sp, sp, #12 @ make space
3
4     mov r3, #1
5     str r3, [sp, #0] @ x is sp+0
6
7     mov r3, #16
8     str r3, [sp, #4] @ y is sp+4
9
10    ldr r3, [sp, #0]
11    str r3, [sp, #8] @ tmp is sp+8
12
13    ldr r3, [sp, #4] @ put y into x
14    str r3, [sp, #0]
15
16    ldr r3, [sp, #8]
17    str r3, [sp, #4]
18
19    add sp, sp, #12 @ shrink stack
20    bx lr

```